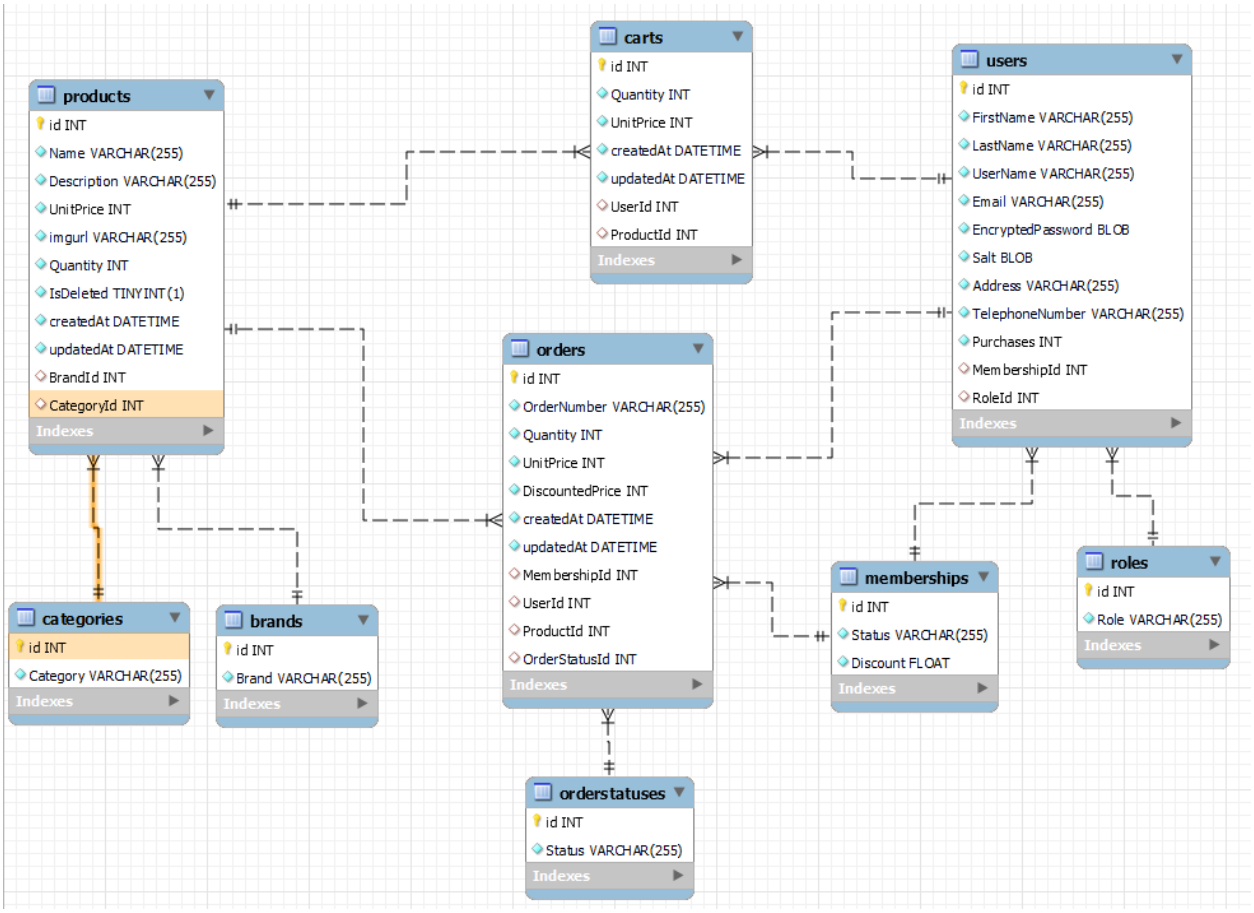
Exam Reflection Report:

Database:



Database Relations:

A Product can only have a single category. A category can be in many products: One to many relation

Products belong to a single brand. A brand has many products: One to many relation

An order row can only have one product. A product can have many orders: One to many relation

A product can be in many carts. A cart can only have one of each product: One to many relation

A cart can only have one user. A user can have a cart with multiple products in it: one to many relation

A user can have one role. A role can have many users: one to many relation

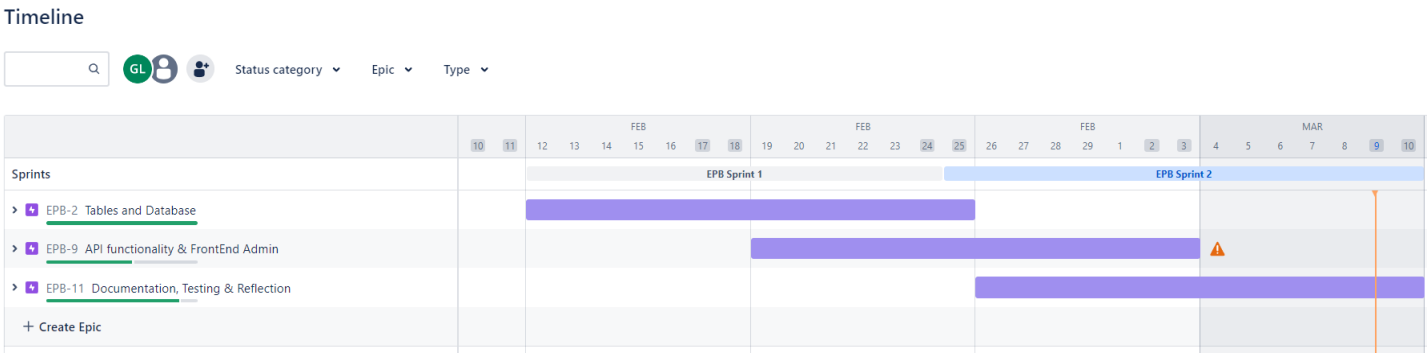
A user can only have one membership level. A membership level can have many users: one to many relation

An order can have one user. A user can have many orders: One to many relation

An order can have one membership level. Membership can be in many orders: One to many

An order can have one order status. Order status can have many orders: One to many.

Jira Roadmap - Epics and Sprints:



Reflection:

I started by diving the project into smaller chunks, that would fit into a sprint each.
I ended up with three different sections.

- Database creation and relations:
- API functionality and Front-end:
- Documentation, testing and reflection:

I think the logic on tables and relations are challenging for me. So, I wanted to focus on it, and get the major tables and relations done before starting the API.
At first, I followed the assignment to find all the obvious tables and relations. When I had all the major tables under control, I started on the API. My reasoning was that I would naturally find where my relations were wrong, or where I needed to update table attributes or where I had to add a completely new table.
At first, I did not have a separate table for roles. My first thought was to simply write the role in the table when creating a new user. But what if we want to change all the role names in the future? Then I would have to change all the roles of each user individually. But if each user points to a role table, then I can change only one variable in the roles table, and have it effect every user in the database.

To test my API along the way I used postman vigorously. Most of the API went relatively pain free. But at the checkout and order functionality I had to really put this brain into overdrive. This is where a lot of connections are made, and values need to change in the right order. User purchases must change, membership status must be calculated, product quantity must be changed.
If a user membership status levels up during a purchase, the next cart has to calculate with the new status discount. At first, I used the req.user to store the user discount on login. But the problem is if the user buys something again without logging out and back in. The req.user values are not updated when the database is. I therefore changed it, so I grab the user from the database every time, that way I got up-to-date status and discount.

For brands and categories, it does not make sense to allow more than one instance of the same name. So, they have a unique name requirement. I therefore choose to delete these based on names and not id. It was more intuitive for me as I was working in Postman.

One thing that confused me midway through the exam was big or small letters on objects. I originally had small first letters for objects coming from req.body, and then I got big first letters on objects from the database. At some point I got confused when creating products. So, to mitigate my confusion, the req.body on products starts with a big letter. I would change this back, but with little time left and fear of causing unforeseen bugs, I am leaving it as is.

As I got done with the general functionality of the API, I started to work on validation on every route. Making sure email had a @ in them, and quantity was a number and so on.

I then went over to the front end and admin pages.
This required a lot of time and effort on my part, as I have not gotten as much training in the frontend since we started JavaScript servers.
But I used my earlier assignments a lot to see what I had done before. I used bootstrap to find tables and navbar and correct syntax. ChatGPT was used to help me with html and bootstrap class syntax as well.

I was very frustrated the first day of the front-end part. It really looked like I was going to struggle with every area. But as I got back into the groove of things and got the navbar functioning and my first/brand site going, it steamrolled quickly, and I got through most of it on day 2. I had only the /product page left, which was the most challenging. But I found a solution which was very intuitive and easy to use, even though I do not think it looks very

pretty. But it gets the job done, no problem. Also added a search bar to /products. Not sure it was required in the frontend, but it was fun to add.

The /init was straight forward, just had to make sure it did not cause any issues if it was run more than once.

Close to the end I started on the swagger and testing. Swagger worked fine without many issues. But most of the syntax I must look up every time. I have not gotten fluent in that language.

Testing also went better this time, compared to my API CA. Felt more in control and got it working quite fast. But IMPORTANT NOTE!!! Running the test once will result in all tests being successful. But running it a second time will not! Because the test is adding a TEST_PRODUCT to the database, and the delete test is only soft deleting it. The next test cycle will not have all tests passed, because TEST_PRODUCT is already in the database. Causing unique issues.

Overall, an incredibly fun exam project. I learn a lot while doing these, and I really like having a week or more to do these assignments. It feels like a proper project i would encounter out in the wild.